TQS: Quality Assurance manual

***André Oliveira [107637]*, *Duarte Cruz*** ***[107359], Tomás Matos [108624], Diogo Almeida* *[108902]***

v2024-04-17

[1 Project management 1](#_Toc132723568)

[1.1 Team and roles 1](#_Toc132723569)

[1.2 Agile backlog management and work assignment 1](#_Toc132723570)

[2 Code quality management 2](#_Toc132723571)

[2.1 Guidelines for contributors (coding style) 2](#_Toc132723572)

[2.2 Code quality metrics 2](#_Toc132723573)

[3 Continuous delivery pipeline (CI/CD) 2](#_Toc132723574)

[3.1 Development workflow 2](#_Toc132723575)

[3.2 CI/CD pipeline and tools 2](#_Toc132723576)

[3.3 System observability 2](#_Toc132723577)

[3.4 Artifacts repository [Optional] 2](#_Toc132723578)

[4 Software testing 2](#_Toc132723579)

[4.1 Overall strategy for testing 2](#_Toc132723580)

[4.2 Functional testing/acceptance 3](#_Toc132723581)

[4.3 Unit tests 3](#_Toc132723582)

[4.4 System and integration testing 3](#_Toc132723583)

[4.5 Performance testing [Optional] 3](#_Toc132723584)

[This report should be written for new members coming to the project and needing to learn what are the QA practices defined. Provide concise, but informative content, allowing other software engineers to understand the practices and quickly access the resources.

Tips on the expected content, along the document, are meant to be removed.

You may use English or Portuguese; do not mix.]

# Project management

## Team and roles

Within our team, we've designated specific roles to ensure smooth operation and efficient progress towards our project goals. André Oliveira takes on the pivotal role of Team Manager. He oversees the entire team's progress, coordinating tasks, managing resources, and ensuring alignment with project objectives. André serves as the central communication hub between the team and stakeholders, deftly resolving any conflicts that may arise.

Duarte Cruz, our DevOps specialist, is instrumental in the development, deployment, and maintenance of our project's infrastructure and tools. His focus on automating processes, ensuring continuous integration and delivery, and optimizing system performance is critical for streamlining our workflow and enhancing reliability.

Tomás Matos holds the crucial position of Product Owner, acting as the voice of our customers. He meticulously defines and prioritizes our product backlog, gathering requirements and ensuring that our deliverables meet and exceed customer expectations. Tomás works closely with the team, guiding them towards fulfilling the project's vision and goals.

Lastly, Diogo Almeida, our Architect, takes charge of designing the project's overall structure and architecture. He formulates technical strategies, selects appropriate technologies, and ensures that our system meets stringent performance, scalability, and security standards. Diogo collaborates closely with the development team, providing architectural guidance and addressing any technical challenges that arise throughout the project lifecycle.

## Agile backlog management and work assignment

In our project, we've embraced agile methodologies for managing our backlog and assigning work, with a strong emphasis on user stories and Behavior-Driven Development (BDD) testing. Our approach revolves around breaking down features into user-centric stories that deliver tangible value to our end-users.

For backlog management, we rely on Jira, a powerful tool that allows us to meticulously organize our backlog. With Jira, we create and prioritize user stories based on factors like customer feedback, business objectives, and technical dependencies. This prioritization ensures that we're always working on the most impactful features first, maximizing value delivery.

Our teams engage in collaborative planning sessions where we collectively decide which user stories to tackle in the upcoming sprint. During these sessions, we break down larger features into smaller, more manageable tasks. This task decomposition makes it easier to assign work to team members based on their expertise and availability.

A key aspect of our work assignment process is the integration of BDD testing. Before any work begins, we define acceptance criteria for each user story in collaboration with stakeholders. These criteria serve as the foundation for writing BDD test scenarios, ensuring that the implemented functionality aligns with the desired outcomes.

By adhering to these agile practices, we aim to maintain a well-organized backlog, efficiently assign work, and deliver high-quality software that meets the needs and expectations of our users.

# Code quality management

## Guidelines for contributors (coding style)

In our project, we've established a set of guidelines known as the "GateMate Coding Style" to maintain consistency and clarity in our codebase. These guidelines are essential for ensuring that all contributors follow a unified approach to coding, making the codebase more readable, maintainable, and collaborative.

The "GateMate Coding Style" encompasses several key aspects. Firstly, it outlines naming conventions for variables, functions, classes, and other identifiers, emphasizing the use of descriptive and meaningful names for improved readability.

Additionally, the coding style defines formatting rules, including guidelines for code indentation, spacing, and line length. Consistent formatting standards make it easier for developers to understand and navigate the code.

Clear and informative comments are another crucial component of our coding style. We stress the importance of documenting code functionality, rationale, and potential pitfalls to aid future maintainers in understanding the codebase.

Moreover, our coding style provides guidelines for organizing code files, modules, and directories in a logical and intuitive manner. This structured approach simplifies code navigation and reduces cognitive overhead for developers.

Incorporating best practices for error handling, exception handling, and code optimization is also part of our coding style. By following established best practices, we ensure that our code is robust, efficient, and scalable.

Comprehensive documentation is required for public interfaces, APIs, and code modules, including usage examples, parameter descriptions, return values, and relevant caveats or considerations.

Lastly, our coding style includes guidelines for version control usage, such as commit message conventions, branching strategies, and code review processes. Adhering to these guidelines promotes collaboration, code quality, and project traceability.

Overall, the "GateMate Coding Style" serves as a foundational framework for effective collaboration and ensures that our software maintains the highest standards of quality and professionalism throughout its development lifecycle.

## Code quality metrics and dashboards

In our project, integrating SonarCloud into our development workflow has significantly elevated our approach to ensuring code quality and minimizing technical debt. We've seamlessly woven SonarCloud into our Continuous Integration (CI) pipeline, making it an integral part of our development process. Before any pull request can be merged into the main branch, it undergoes rigorous automated code analysis by SonarCloud.

Our adherence to SonarCloud's default quality gates is unwavering. We've set a stringent policy dictating that all code changes must pass these quality gates before they're deemed fit for merging into the main branch. This ensures that only the highest quality code, free from major issues and violations, makes its way into our production environment.

When SonarCloud flags issues or violations during the analysis process, we have a standardized workflow in place for addressing them. Developers take ownership of resolving any issues highlighted by SonarCloud in their respective pull requests before they're eligible for merging. This commitment to resolving issues in a timely manner ensures that code quality remains a top priority throughout the development lifecycle.

During code reviews, SonarCloud's reports and dashboards serve as valuable reference points. Reviewers rely on SonarCloud's findings to identify potential areas of improvement or to discuss any code quality issues with the author. This collaborative approach fosters a culture of continuous improvement and collective responsibility for code quality within our team.

Regular analysis of historical trends in code quality metrics using SonarCloud's dashboards allows us to gain insights into our development practices over time. Tracking these trends enables us to pinpoint areas for improvement, measure the impact of our code quality initiatives, and make informed decisions to enhance our development processes further.

We're committed to providing training and support to developers to help them understand SonarCloud's findings and address code quality issues effectively. This proactive approach empowers developers to write cleaner, more maintainable code and instills a sense of accountability for code quality within the team.

By integrating SonarCloud into our development process in these ways, we've been able to uphold a high standard of code quality, reduce technical debt, and deliver software that's more reliable and maintainable for our users.

# Continuous delivery pipeline (CI/CD)

## Development workflow

In our project, we've embraced a Git workflow, akin to the GitHub Flow model, to orchestrate our development process smoothly. Here's a rundown of how our workflow operates and its alignment with user stories:

Our Git workflow starts with developers creating feature branches derived from the main branch. These branches are dedicated to specific user stories or features identified in our backlog. Developers work diligently within these branches to implement the functionality outlined in the corresponding user story.

Once a developer completes work on a feature or user story, they initiate a pull request (PR) to merge their changes back into the main branch. This pull request becomes the focal point for code review and collaboration among team members.

During the code review phase, team members thoroughly assess the code changes introduced in the pull request. They scrutinize the code for adherence to coding standards, correctness, and overall quality. Feedback and suggestions for improvement are provided, fostering a culture of collaboration and continuous improvement.

Our CI pipeline is triggered automatically upon opening a PR. This pipeline runs a battery of tests, including unit tests, integration tests, and static code analysis using SonarCloud. The code changes must pass all checks in the CI pipeline before they're considered ready for merging into the main branch.

Once the code changes have undergone successful code review and have passed all checks in the CI pipeline, they are merged into the main branch. This integration marks the completion of the user story, making the implemented functionality part of the project's codebase.

Each feature branch corresponds directly to a specific user story or feature identified in our backlog. Developers work within these branches to implement the functionality outlined in the user stories, ensuring a clear mapping between code changes and user requirements.

In terms of code review practices, our team emphasizes thoroughness and collaboration. Reviewers provide constructive feedback and suggestions for improvement during the code review process. We leverage tools like SonarCloud for automated static code analysis, which helps identify potential code quality issues early in the review process.

Our team's Definition of Done (DoD) for a user story encompasses several criteria. These include successful implementation of the functionality, thorough code review, passing of automated tests, seamless integration into the main branch, and any necessary documentation updates.

By adhering to this workflow and code review practices, we maintain an efficient, collaborative development process focused on delivering high-quality software that meets the needs of our users.

## CI/CD pipeline and tools

[Description of the practices defined in the project for the continuous integration of increments and associated resources. Provide details on the tools setup and config.]

[Description of practices for continuous delivery, likely to be based on *containers*]

## System observability

## Artifacts repository [Optional]

[Description of the practices defined in the project for local management of Maven *artifacts* and associated resources. E.g.: a[rtifactory](https://www.jfrog.com/artifactory/)]

# Software testing

## Overall strategy for testing

In our project, we've crafted a robust testing strategy that encompasses various methodologies and tools to ensure comprehensive test coverage and uphold code quality. At the core of our approach lies Test-Driven Development (TDD), where developers write tests for new features or changes before implementing the corresponding code. This proactive method ensures that code is thoroughly tested from the outset, fostering better design practices and minimizing the occurrence of bugs.

In addition to TDD, we've embraced Behavior-Driven Development (BDD) principles, leveraging tools like Cucumber to describe and test the behavior of our software from an end-user perspective. Cucumber enables us to write executable specifications in plain language, promoting collaboration with stakeholders and ensuring alignment with business requirements.

Our testing arsenal comprises a mix of different tools tailored to address various testing needs within our project. For instance, we utilize REST-Assured for API testing, Selenium for automated browser testing, and JUnit for unit testing Java code. This diverse toolset empowers us to test different layers of our application effectively, ensuring that each component behaves as expected.

Furthermore, our testing strategy is tightly integrated into our Continuous Integration (CI) process. Automated tests are executed automatically upon code changes, leveraging CI platforms like Jenkins or GitHub Actions. This seamless integration provides rapid feedback to developers and safeguards against regressions, ensuring that code changes maintain the expected level of quality.

In our CI process, test results play a pivotal role. We have established policies and practices to meticulously monitor and consider test results before merging code changes into the main branch. Pull requests must pass a battery of automated tests, including unit tests, integration tests, and acceptance tests, before they are approved for merging. Any test failures or regressions prompt immediate investigation and resolution by developers.

Overall, our testing strategy emphasizes proactive testing, collaboration, and automation to deliver high-quality software that meets the needs of our users. By embedding testing into our development process and CI pipeline, we ensure that code changes undergo rigorous validation, mitigating the risk of defects and enhancing the overall reliability of our software.

## Functional testing/acceptance

In our project, our approach to functional testing revolves around the user stories, ensuring that each test scenario aligns closely with the intended user interactions and workflows within the application. We've established a structured policy that prioritizes the validation of functionality from the user's perspective, leveraging various resources and practices to achieve this goal.

At the core of our functional testing strategy is the definition of clear and comprehensive test scenarios derived directly from user stories. These scenarios serve as the foundation for our tests, encapsulating the various paths and behaviors that users are expected to follow while interacting with the application. By focusing on user stories, we ensure that our tests are closely aligned with the requirements and expectations outlined by stakeholders.

To implement these tests effectively, we rely on an automation framework that allows us to automate repetitive test cases and execute them consistently. This framework streamlines the testing process, enabling us to validate the functionality of the application efficiently and reliably across different environments.

Managing test data is another crucial aspect of our functional testing strategy. We carefully curate test data sets that represent diverse user profiles, input parameters, and edge cases, ensuring that our tests accurately reflect real-world scenarios. This approach enhances the thoroughness and effectiveness of our testing efforts, uncovering potential issues that users may encounter in practical usage scenarios.

Integration into our Continuous Integration (CI) pipeline ensures that functional tests are executed automatically upon code changes. This integration enables us to detect regressions early in the development process, preventing the introduction of bugs and ensuring that new features or changes do not compromise the functionality of the application.

Following each execution cycle, we generate detailed test reports that provide insights into the status of each test case and any failures encountered. These reports facilitate collaboration among team members, enabling developers, testers, and stakeholders to identify and address issues promptly.

Incorporating functional tests into our regression testing strategy ensures that existing functionality remains intact as new features are introduced or changes are made. By prioritizing user stories in our functional testing efforts, we uphold the quality and reliability of our software, ultimately delivering an exceptional user experience.

## Unit tests

In our project, we have a clear and structured policy for writing unit tests, focusing on validating individual components of our software from a developer's perspective. These tests are designed to ensure that each unit of code behaves as expected and meets the specified requirements, contributing to the overall quality and maintainability of our codebase.

Our approach to unit testing encompasses several key elements. Firstly, we define a target level of test coverage for our codebase, providing a guideline for developers to ensure critical components and edge cases are adequately covered by tests.

We also advocate for Test-Driven Development (TDD) practices, encouraging developers to write unit tests before implementing the corresponding code. This approach promotes better design, increases code maintainability, and ensures thorough testing from the outset of development.

To isolate units of code for testing, we employ techniques such as mocking and stubbing to simulate the behavior of dependencies. This allows us to test components in isolation, without relying on external dependencies or infrastructure, facilitating faster and more focused testing efforts.

We leverage industry-standard test frameworks and tools, such as JUnit to facilitate the writing and execution of unit tests. These frameworks provide essential functionalities for defining test cases, executing tests, and reporting test results.

Unit tests are integrated into our Continuous Integration (CI) pipeline, where they are executed automatically upon code changes. This integration ensures that any changes introduced by developers do not break existing functionality and that the codebase remains stable and maintainable over time.

During code reviews, unit tests play a crucial role, with developers reviewing each other's tests to ensure adequate coverage of intended functionality and edge cases. This collaborative approach promotes knowledge sharing and ensures the robustness and effectiveness of our tests.

Unit tests provide valuable support during the refactoring process, acting as a safety net to identify and address any regressions introduced by code changes. By adhering to these practices, we ensure that our unit testing efforts are systematic, effective, and contribute significantly to the overall quality and maintainability of our codebase.

## System and integration testing

In our project, we have a clear strategy for system and integration testing, particularly focusing on API testing. Developers create and maintain integration tests alongside their code changes, ensuring thorough validation of API functionality. We leverage automation frameworks like REST-Assured or Spring MVC Test to automate test execution, ensuring consistency and efficiency. Test data is carefully managed to reflect real-world scenarios. Integration tests are seamlessly integrated into our CI pipeline for automatic execution, and detailed reports provide insights into test status and failures for prompt resolution. This approach ensures the reliability and performance of our APIs, delivering a seamless experience to users.

## Performance testing [Optional]

[Project policy for writing performance tests and associated resources.]